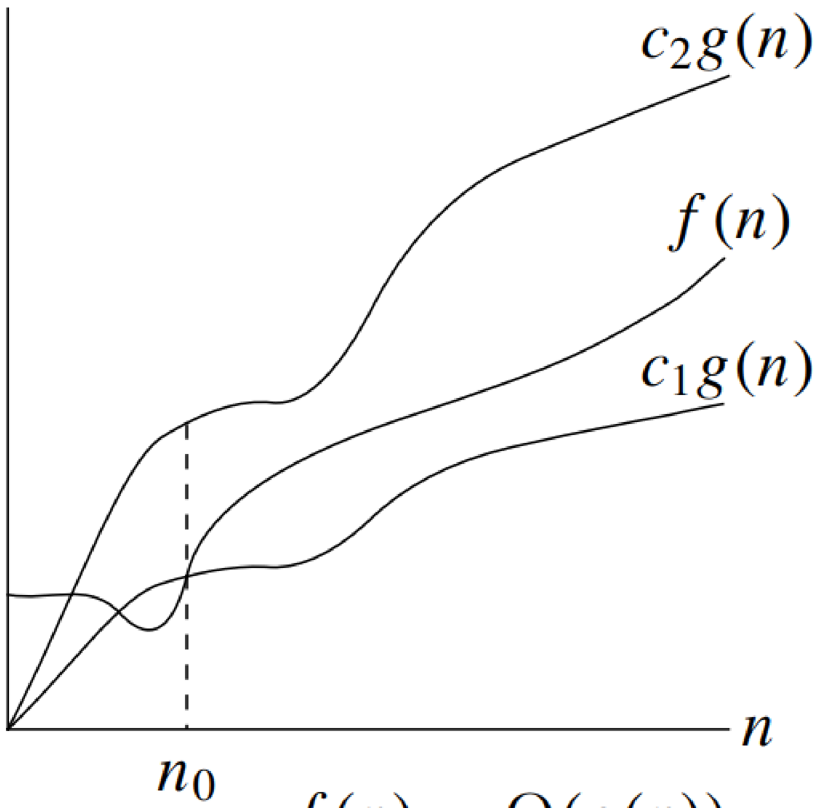
For a given function $g(n)$, we denote by $\Theta(g(n))$ the *set of functions*

$$\Theta(g(n)) = \{f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ for all } n \ge n_0\}\ .^1$$

$\Theta$ notation (asymptotically tight bound for $f(n)$)




$f(n) = \Theta(g(n))$

# Example:

$$f(n) = \frac{1}{2}n^2 - 3n = \Theta(n^2)$$

To justify above statement, must determine positive constant $c_1$, $c_2$, and $n_0$ such that:

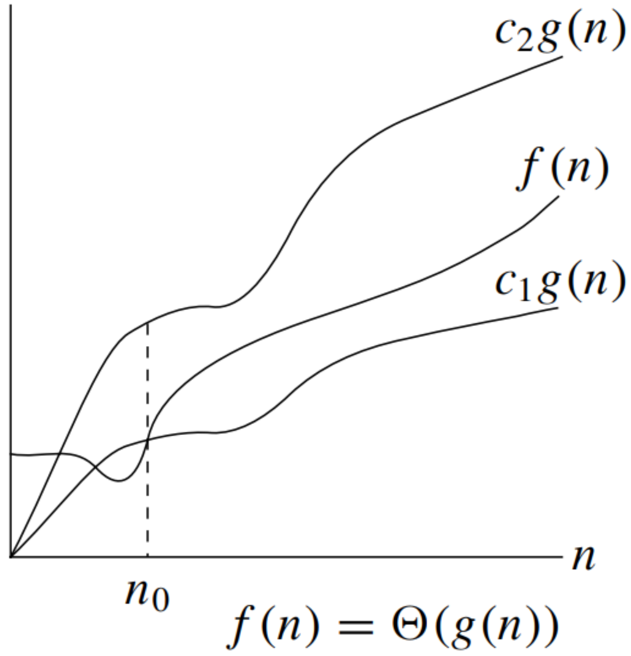
$$c_1 n^2 \le \frac{1}{2}n^2 - 3n \le c_2 n^2 \text{ for all } n \ge n_0$$

Dividing by $n^2$ yields:

$$c_1 \le \frac{1}{2} - \frac{3}{n} < c_2$$

rhs inequality holds for any value $n \ge 1$ by choosing $c_2 \ge \frac{1}{2}$

Likewise, lhs inequality holds for any value $n \ge 7$ by choosing $c_1 \le \frac{1}{14}$.




Thus, by choosing $c1 = \frac{1}{14}$, $c_2 = \frac{1}{2}$ and $n_0 = 7$, can verify that $\frac{1}{2}n^2 - 3n = \Theta(n^2)$

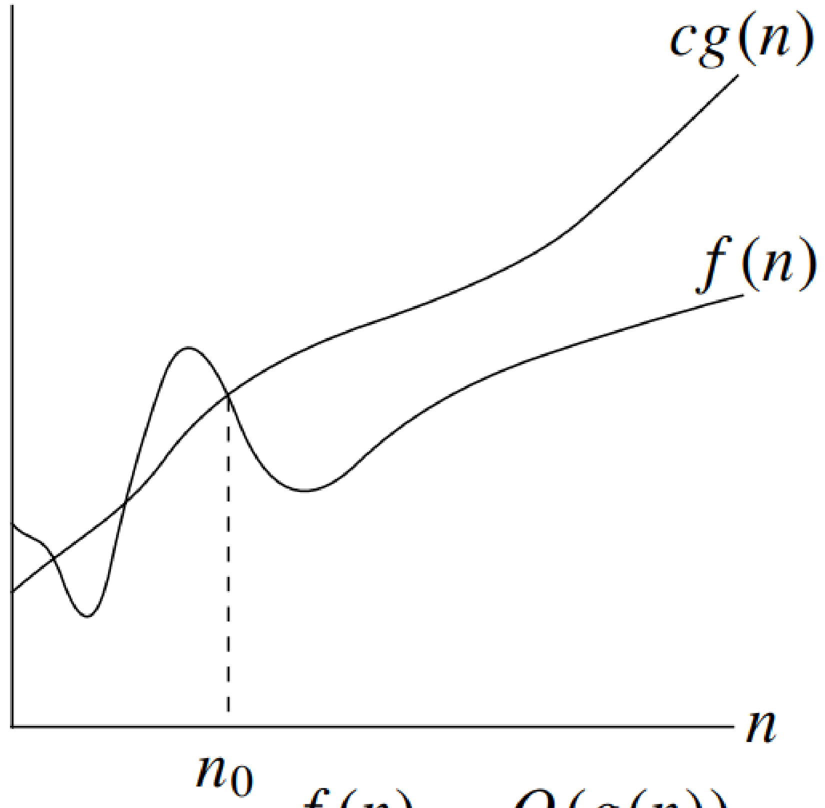
For a given function $g(n)$, we denote by $O(g(n))$ (pronounced "big-oh of $g$ of $n$" or sometimes just "oh of $g$ of $n$") the set of functions

$$O(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0\} .$$

$O$ notation (asymptotic *upper* bound for $f(n)$).

Also,

$$\Theta(g(n)) \subseteq O(g(n))$$




$f(n) = O(g(n))$

# $O$ notation

- $O$ notation gives an asymptotic upper bound on a function, to within a constant factor
- When $O$ notation used to bound worst-case running time of an algorithm, we have a bound on the running time of the algorithm for *any* input
- Any quadratic function is in $O(n^2)$
- Also, any *linear* function $an + b$ is in $O(n^2)$
  - When we write $f(n) = O(g(n))$ we are merely claiming that some constant multiple of $g(n)$ is an asymptotic upper bound on $f(n)$ with no claims about how tight the bound may be
- Can often describe running time of algorithm using $O$ notation by inspecting algorithm's overall structure (next slide)

# Analyzing insertion sort at a glance

- Doubly nested loop structure
- Cost of each iteration of inner loop is $O(1)$ (doesn't depend on size of $A$)
- Indices $i$ and $j$ are both at most $n$
- Inner loop is executed at most once for each of the $n^2$ pairs of values for $i$ and $j$

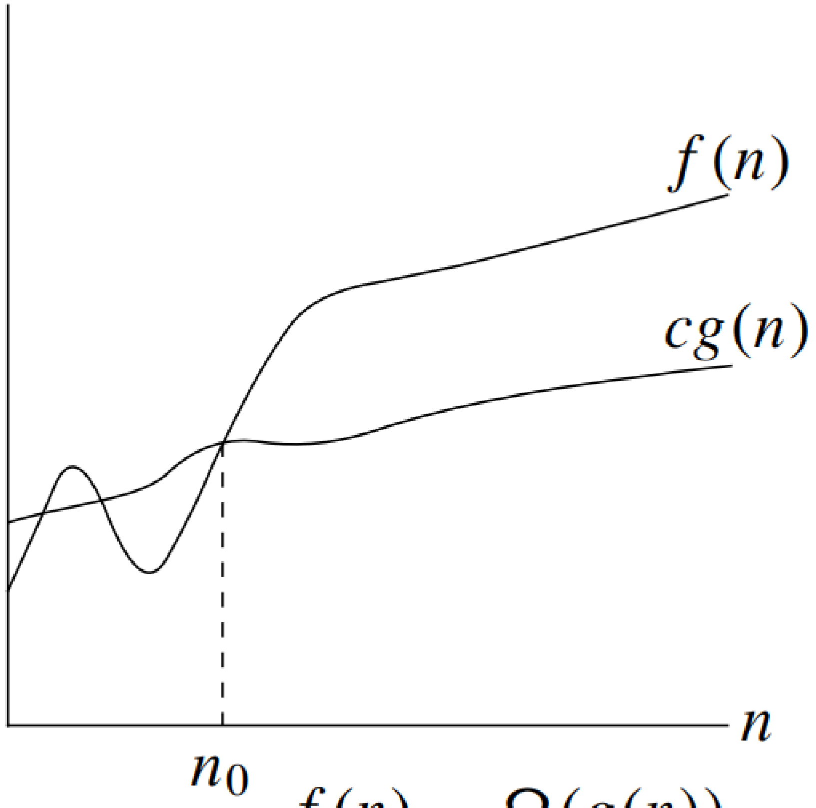
INSERTION-SORT$(A)$

```
for j ← 2 to length[A]
    do key ← A[j]
       ▷ Insert A[j] into the sorted sequence A[1 .. j − 1].
       i ← j − 1
       while i > 0 and A[i] > key
           do A[i + 1] ← A[i]
              i ← i − 1
       A[i + 1] ← key
```

For a given function $g(n)$, we denote by $\Omega(g(n))$ (pronounced "big-omega of $g$ of $n$" or sometimes just "omega of $g$ of $n$") the set of functions

$$\Omega(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \le cg(n) \le f(n) \text{ for all } n \ge n_0\} .$$

# $\Omega$ notation (asymptotic *lower* bound for $f(n)$)




$f(n) = \Omega(g(n))$

| Function | Name |
|---|---|
| $c$ | Constant |
| $\log N$ | Logarithmic |
| $\log^2 N$ | Log-squared |
| $N$ | Linear |
| $N \log N$ | |
| $N^2$ | Quadratic |
| $N^3$ | Cubic |
| $2^N$ | Exponential |

**Figure 2.1** Typical growth rates